弹性微服务

实践教程

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

实践教程
托管静态网站
弹性微服务应用访问公网（通过 NAT 网关）
弹性微服务应用访问公网（通过 API 网关）
弹性微服务应用运行失败故障排查指南
使用 API 网关快速访问弹性微服务应用
Java 应用调优实践教程
Java 8 迁移至 Java 11 实践教程

版权所有：腾讯云计算（北京）有限责任公司

实践教程
托管静态网站

最近更新时间：2024-06-14 18:19:51

操作场景

弹性微服务通过 **应用实例+CFS** 的方式提供静态网站资源托管的能力，本文以业界流行的静态网站服务 Hugo 为例，为您提供静态资源托管的实践教程。下述将通过 Hugo 生成个人静态博客，然后通过弹性微服务部署反向代理应用，配合 CFS 管理静态资源，最后通过弹性微服务提供的访问配置，在公网上提供对个人静态博客的访问。

整体流程如下：

1. 本地通过 Hugo 生成静态博客
2. 上传静态博客内容到 CFS
3. 在弹性微服务上部署 nginx 应用并关联 CFS
4. 在弹性微服务上配置 nginx 的网络访问
5. (可选) 配置域名
6. (可选) 配置防火墙
7. (可选) 配置 CDN

操作步骤

步骤1：本地通过 Hugo 生成静态博客

1. 安装 Hugo（参考 Install Hugo）。
   以 MacOS 系统为例，通过如下命令安装：

```
brew install hugo
```

2. 执行如下命令创建静态站点。

```
hugo new site quickstart
```

3. 执行如下命令添加主题。

```
cd quickstart
git init
git submodule add https://github.com/theNewDynamic/gohugo-theme-ananke.git themes/ananke
echo theme = \"ananke\" >> config.toml
```

4. 执行如下命令添加博客。

```
hugo new posts/my-first-post.md
```

5. 执行如下命令构建静态页面。

```
hugo -D
```

6. 生成的静态内容存放在 quickstart 项目的 `public/` 目录下。

```
➜  quickstart git:(master) ✗ ll
total 8
drwxr-xr-x   3 zhangyifei  staff    96B Jul 11 15:45 archetypes
-rw-r--r--   1 zhangyifei  staff    99B Jul 11 15:47 config.toml
drwxr-xr-x   3 zhangyifei  staff    96B Jul 11 15:47 content
drwxr-xr-x   2 zhangyifei  staff    64B Jul 11 15:45 data
drwxr-xr-x   2 zhangyifei  staff    64B Jul 11 15:45 layouts
drwxr-xr-x  11 zhangyifei  staff   352B Jul 11 15:50 public
drwxr-xr-x   3 zhangyifei  staff    96B Jul 11 15:47 resources
drwxr-xr-x   2 zhangyifei  staff    64B Jul 11 15:45 static
drwxr-xr-x   3 zhangyifei  staff    96B Jul 11 15:46 themes
➜  quickstart git:(master) ✗ tree public
public
├── 404.html
├── ananke
│   └── css
│       ├── main.css.map
│       └── main.min.css
├── categories
│   ├── index.html
│   └── index.xml
├── images
│   └── gohugo-default-sample-hero-image.jpg
├── index.html
├── index.xml
├── posts
│   ├── index.html
│   ├── index.xml
│   ├── my-first-post
│   │   └── index.html
│   └── page
│       └── 1
│           └── index.html
├── sitemap.xml
└── tags
    ├── index.html
    └── index.xml

9 directories, 15 files
```

步骤2：上传静态博客内容到 CFS

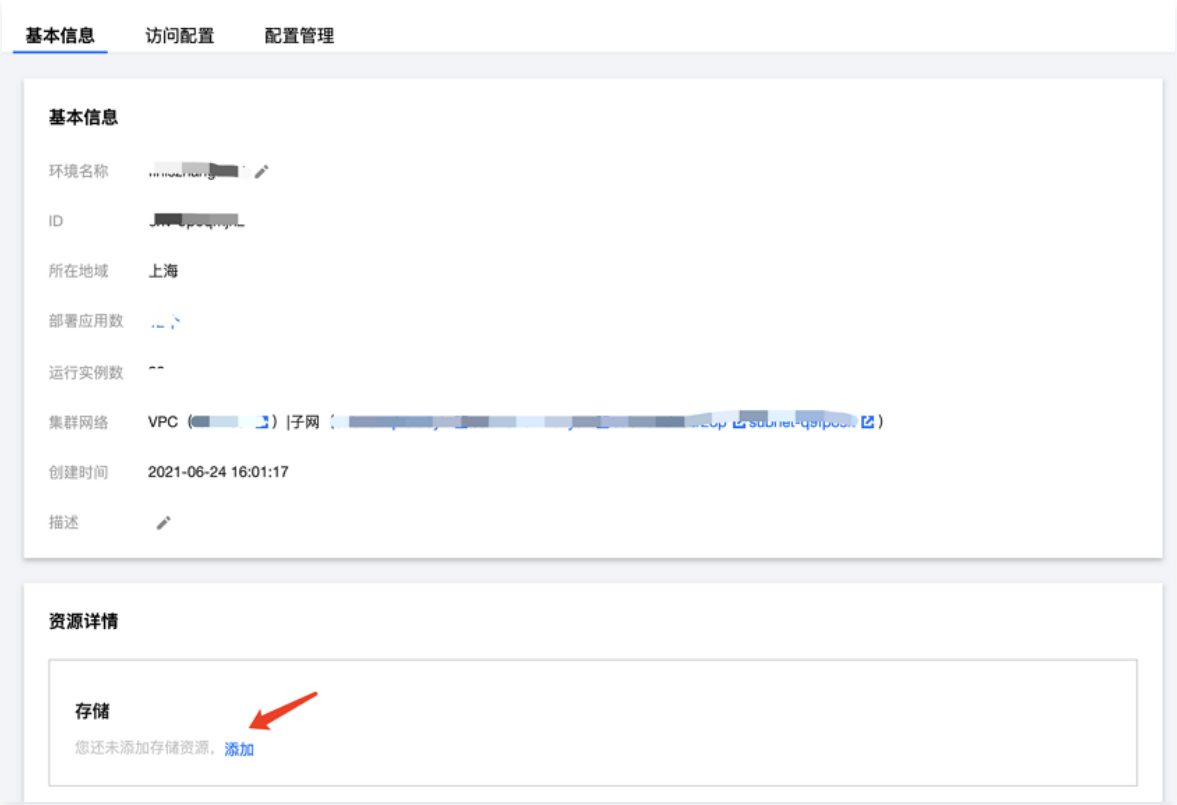
1. 参考 CFS：创建文件系统及挂载点 购买 CFS 文件系统。

> **注意**
> 购买的 CFS 文件系统的**区域**、**VPC 网络** 应与在弹性微服务上部署应用的区域保持一致。

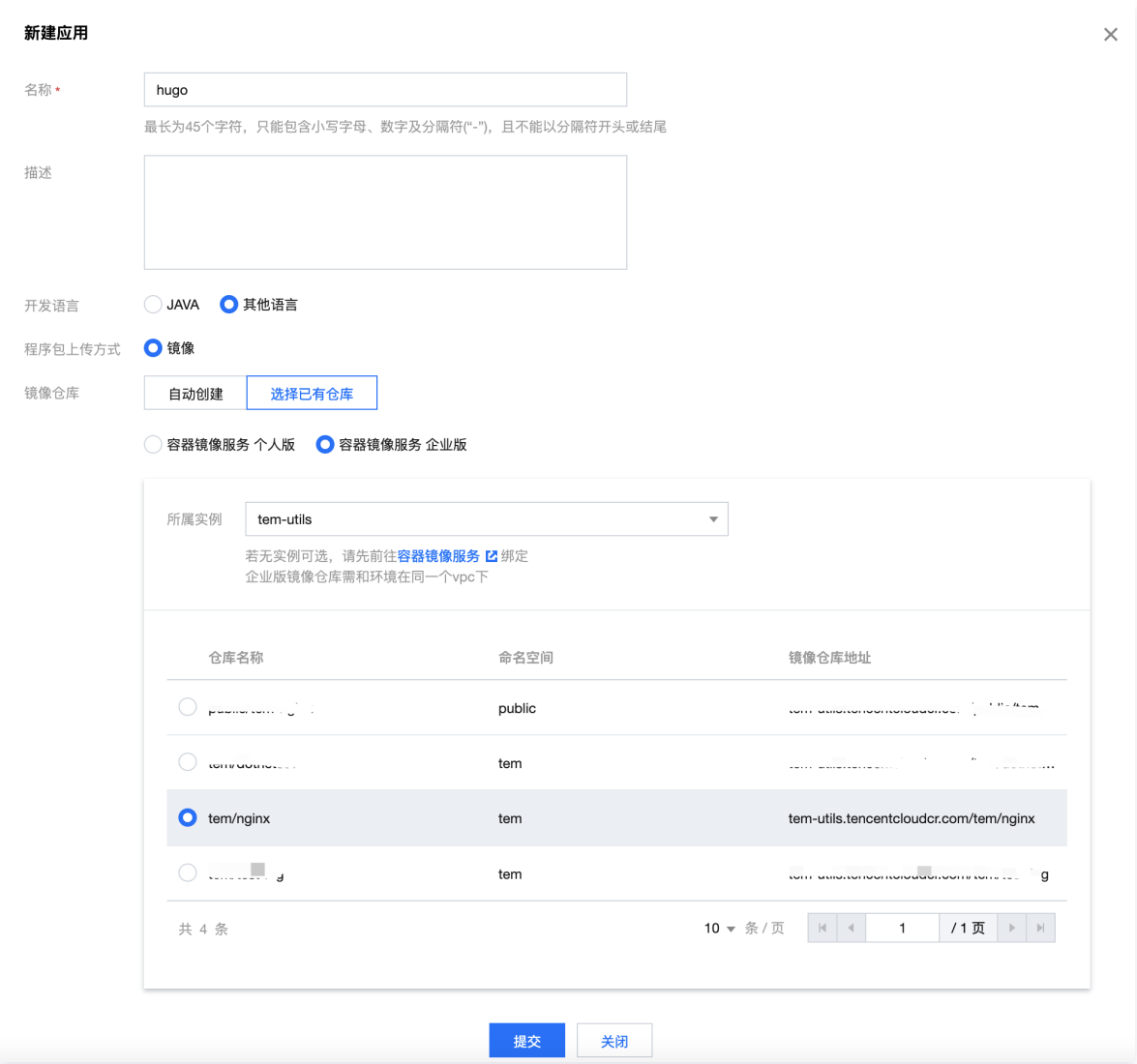
2. 参考 CFS：在 Linux 客户端上使用 CFS 文件系统 或 CFS：在 Windows 客户端上使用 CFS 文件系统 将 hugo 生成的 `public/` 目录中的文件上传到 CFS 文件系统根目录或子目录中。

步骤3：在弹性微服务上部署 nginx 应用并关联 CFS

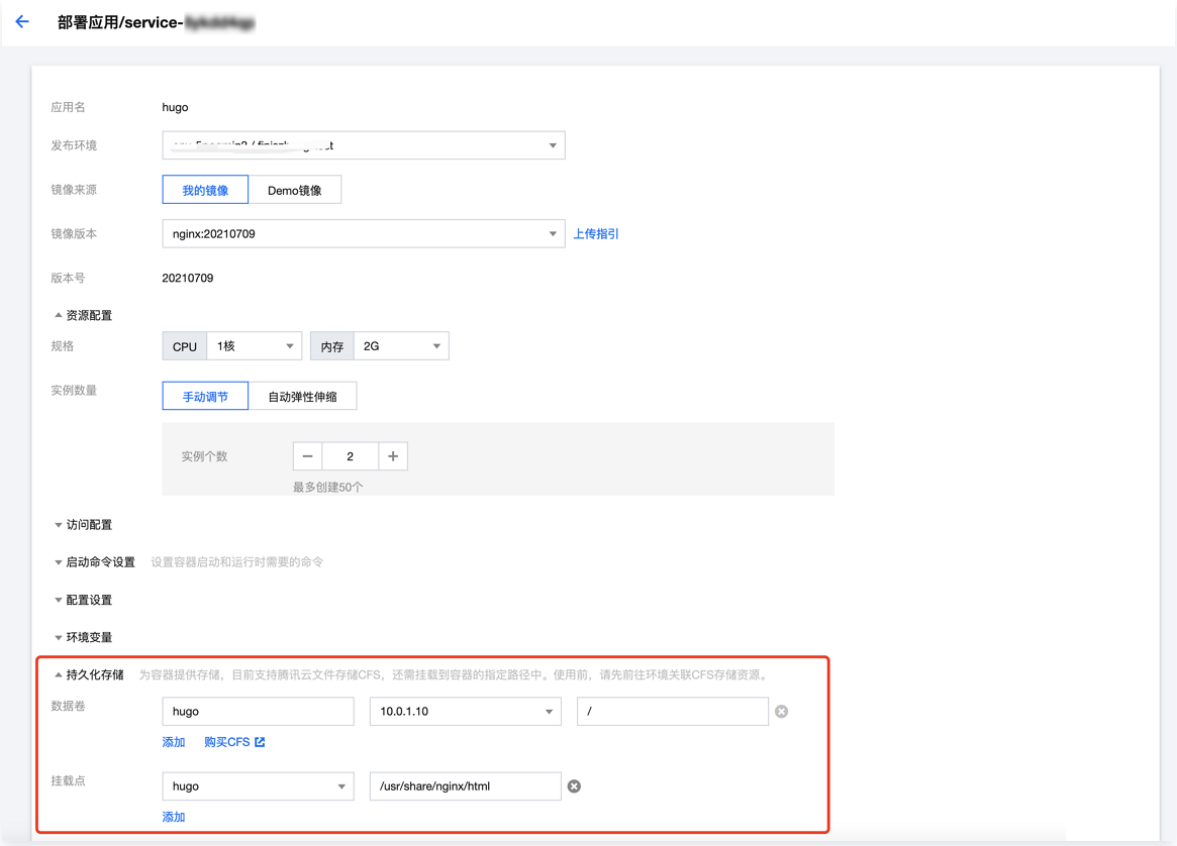
1. 登录 弹性微服务控制台，在应用部署所在的环境中关联上述购买的 CFS 实例。



2. 在 **应用管理** 页面创建一个名为 hugo 的应用。



3. 部署应用，在**持久化存储**模块选择已关联的 CFS 存储资源。

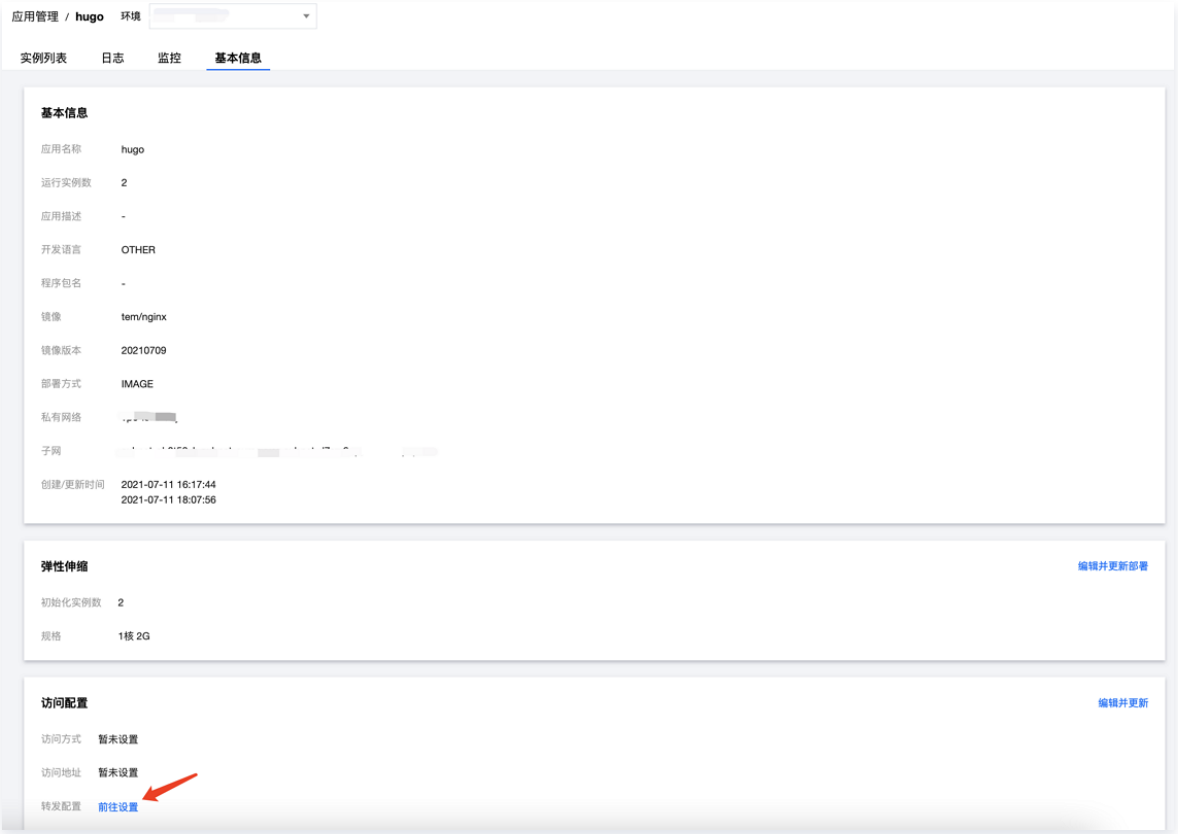


步骤4：在弹性微服务上配置 nginx 的网络访问

方案一：配置转发规则（推荐）

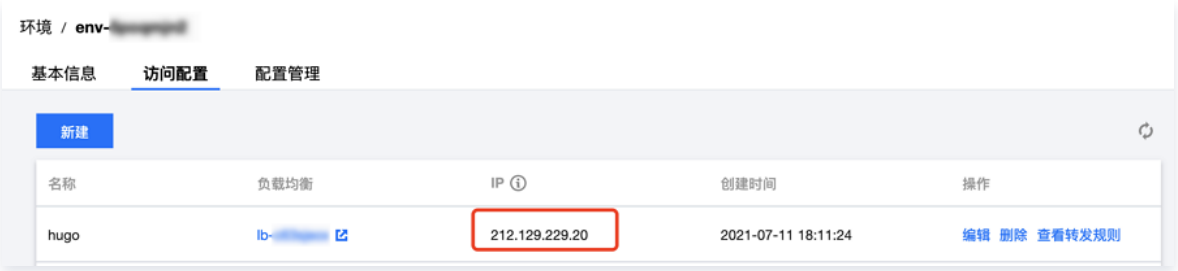
1. 在 应用管理 页面，单击刚刚创建的应用的“ID”，进入应用基本信息页面。

2. 在应用基本信息页面，在**访问配置**模块单击**前往配置**，进入环境访问配置页面。

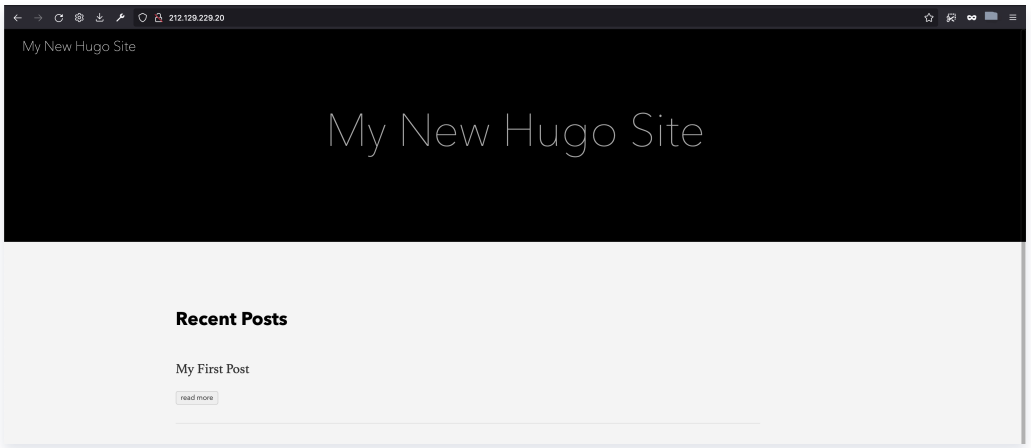


3. 在环境访问配置页面，单击**新建**，创建一条访问配置规则。

4. 单击**完成**，可获取如下 IP 地址。

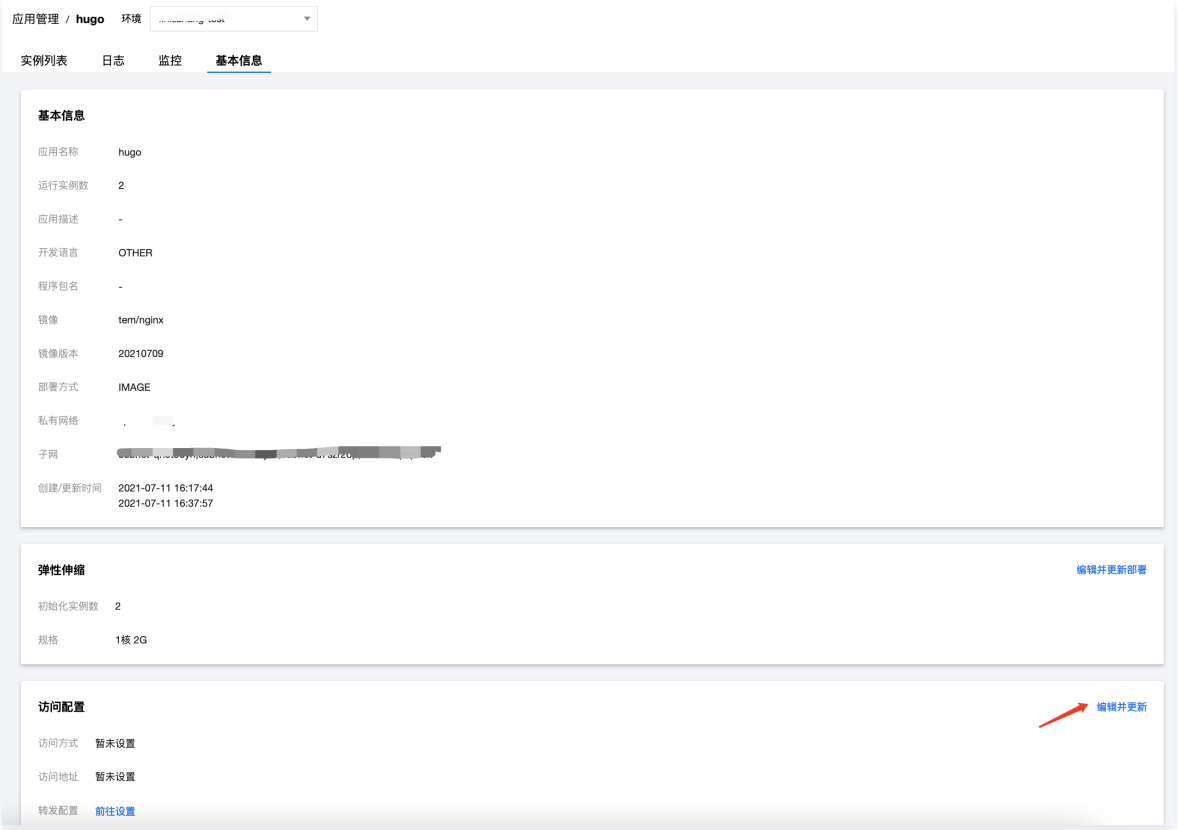


通过生成的地址对 hugo 服务进行访问：



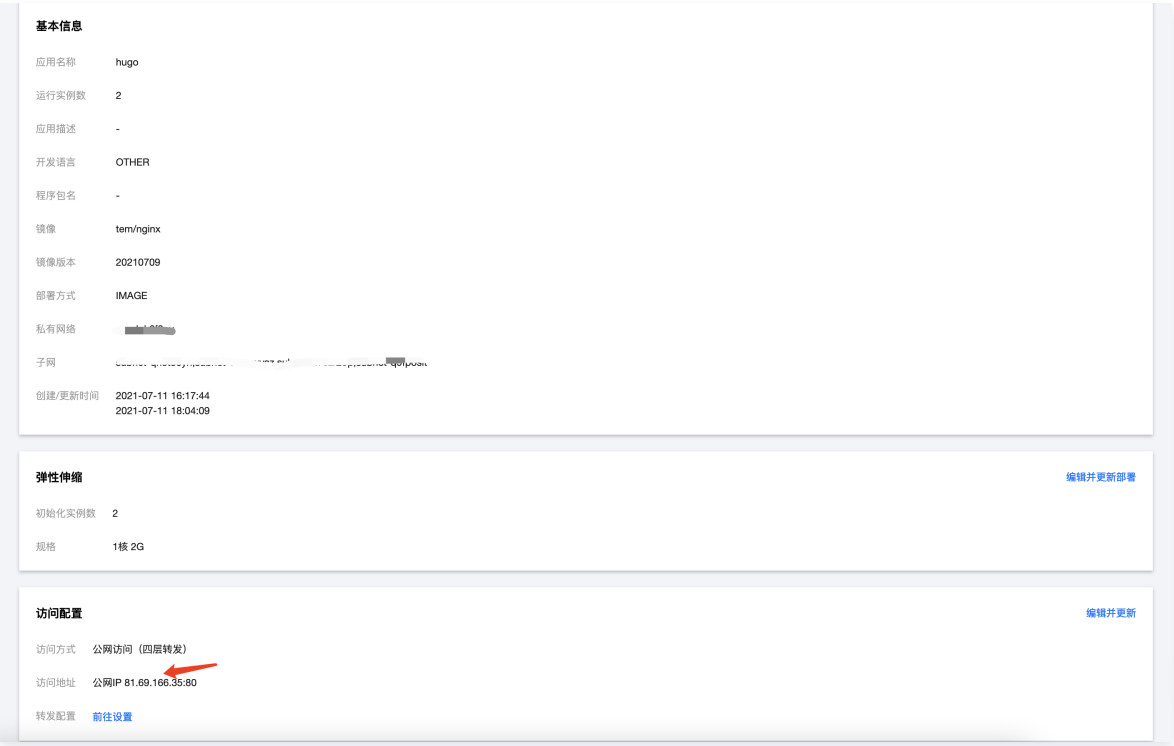
方案二：配置公网 CLB

1. 在 应用管理 页面，单击刚刚创建的应用的“ID”，进入应用基本信息页面。
2. 在应用基本信息页面，在**访问配置**模块单击右上角的**编辑并更新**，添加公网 CLB。

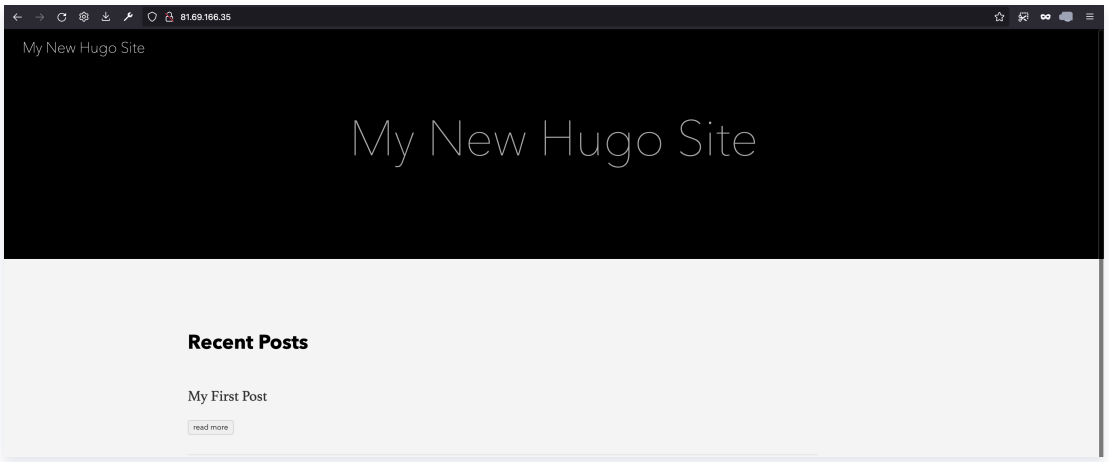


3. 选择公网访问方式，填写端口映射关系。

4. 单击**提交**后，可获取如下公网访问IP。



通过生成的地址对 hugo 服务进行访问：



步骤5：(可选) 配置域名

1. 参考 域名注册 注册域名。
2. 将域名与上述生成的 CLB 实例进行关联，关联后即可通过域名访问静态网站，参考 配置 DNS 解析 DNSPod 将域名指向 CLB 。

步骤6：(可选) 配置防火墙

若静态网站的访问是通过 **转发配置** 设置，则可与腾讯云的 Web 应用防火墙 结合，对静态网站进行安全保护。详细配置可参见 配置 WAF 对负载均衡的监听域名进行 Web 安全防护 。

步骤7：(可选) 配置 CDN

静态网站为了给用户更好的体验，通常会与 CDN 结合提供访问加速，托管的静态网站可与腾讯云 CDN 结合，参见 腾讯云 CDN 新手指引 。

版权所有：腾讯云计算（北京）有限责任公司

弹性微服务应用访问公网（通过 NAT 网关）

最近更新时间：2024-02-01 10:08:21

操作场景

运行在弹性微服务上的应用，由于业务需求等原因，通常会有访问公网的需求。同时，小程序等场景中通常会有访问白名单的需求，此时要求访问公网的应用有固定的公网 IP。
基于上述需求，本文档将会详细介绍部署在弹性微服务上的应用访问公网的方法。

> **说明：**
> 环境已支持一键开启公网访问出口：弹性微服务提供了以环境维度帮助用户开启和管理公网访问入口的能力，以简化原本较为繁琐的网络配置流程，建议参见 在环境中开启公网访问 。

整体思路

弹性微服务中的应用部署在用户的环境中，环境和用户的 VPC 关联，即弹性微服务中的应用本质上是部署在用户的 VPC 内。可通过为 VPC 配置 NAT 网关，并为 NAT Gateway 关联 EIP，实现 VPC 内的应用访问公网。

操作流程

1. 部署弹性微服务应用
2. 创建 NAT 网关实例
3. 在 VPC 控制台中配置 NAT 网关信息
4. 验证公网访问
5. (可选)查询访问公网的 IP

操作步骤

步骤1：部署弹性微服务应用

参考 创建环境 和 创建应用 在弹性微服务控制台中部署应用。

步骤2：创建 NAT 网关实例

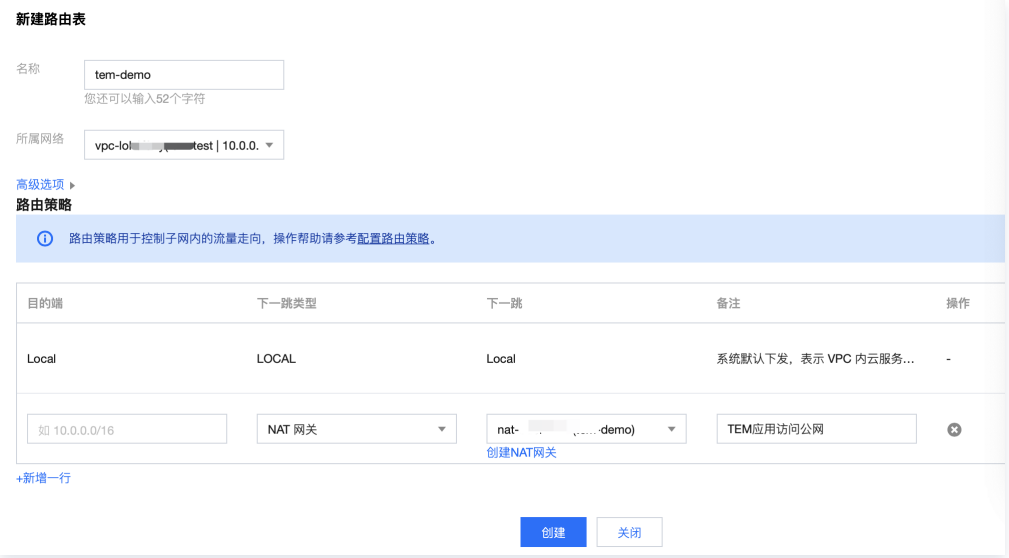
登录 NAT 网关控制台 ，在左侧导航栏选择 **NAT 网关**，选择弹性微服务应用所在的地域，单击**新建**购买 NAT 网关实例。

- **所属网络**：与弹性微服务应用所在环境的 VPC 保持一致。
- **弹性IP**：若没有可用的公网弹性 IP (EIP)，一键跳转进行购买，购买后在 NAT 网关购买页面中刷新即可查到。

步骤3：在 VPC 控制台中配置 NAT 网关信息

1. 在弹性微服务控制台 **环境** 页面单击弹性微服务应用所在的环境卡片，进入环境基本信息页面。
2. 在集群网络处单击 VPC，跳转至 VPC 网络基本信息页面。

3. 单击**路由表**模块，进入路由表配置页面。
4. 在路由表页面，单击**新建**，配置路由表。



- **目的端**：选择需访问的外网 IP 地址，支持配置 CIDR，例如，填写 0.0.0.0/0 会转发所有流量到 NAT 网关。
- **下一跳类型**：选择 NAT 网关。
- **下一跳**：选择**步骤2**中创建的 NAT 网关。
  详细配置参考 自定义路由表 。

5. 在创建好的路由表操作栏单击**更多** > **关联子网**，在关联的子网中选择弹性微服务应用所在环境关联的子网。

步骤4：验证公网访问

1. 在弹性微服务控制台的 **应用管理** 页面单击弹性微服务应用的“ID”，进入应用实例列表页面。
2. 单击应用实例操作栏的 **Webshell**，进入到 Webshell 中。

3. 验证是否可以访问公网。

步骤5：查询访问公网的 IP(可选)

1. 在弹性微服务控制台 **环境** 页面单击弹性微服务应用所在的环境卡片，进入环境基本信息页面。
2. 在集群网络处单击 VPC，跳转至 VPC 网络基本信息页面。

3. 单击 **NAT 网关**模块，进入NAT 网关列表页面。
4. 单击目标 NAT 网关的“ID”，进入 NAT 网关基本信息页面，选择**关联弹性 IP** 页签，可获取访问公网的 IP。

额外费用

使用 NAT 网关 和 EIP 产生额外费用，定价详情参见：

- NAT 网关定价
- EIP 定价

弹性微服务应用访问公网（通过 API 网关）

最近更新时间：2024-01-26 09:58:21

操作场景

运行在弹性微服务上的应用，由于业务需求等原因，通常会有访问公网的需求。而许多场景下，都是这些请求都是 HTTP/HTTPS 请求。您可以使用 API 网关，轻松的通过简单的配置，访问公网的 HTTP/HTTPS 请求。

> **说明**
> 如您的访问公网需求不只包含 HTTP/HTTPS 时，请参见 弹性微服务应用访问公网 来通过配置 NAT 网关实现。

前提条件

请先完成 环境创建 和 创建并部署应用 。

操作步骤

步骤1：在 API 网关中关联公网 HTTP/HTTPS 请求

1. 前往 API网关控制台 ，在左侧导航栏，单击**服务**，进入服务列表页。
2. 选择与部署弹性微服务应用相同的地域，单击页面左上角的**新建**，新建一个服务。
   新建服务时，前端类型可选择 HTTP、HTTPS、HTTP 与 HTTPS 任一种，访问模式选择 VPC 内网，实例类型选择共享型。（关于实例类型的选择，请参见 API 网关文档 实例选择指南 ）

3. 单击 API 网关服务 ID 进入 API 管理页面。单击**新建 API**。
4. 在前端配置中填写 API 名称，前端类型选择 HTTP&HTTPS，路径为“/”，请求方法选择 ANY 以包含所有请求，鉴权类型选择“免认证”，单击**下一步**。

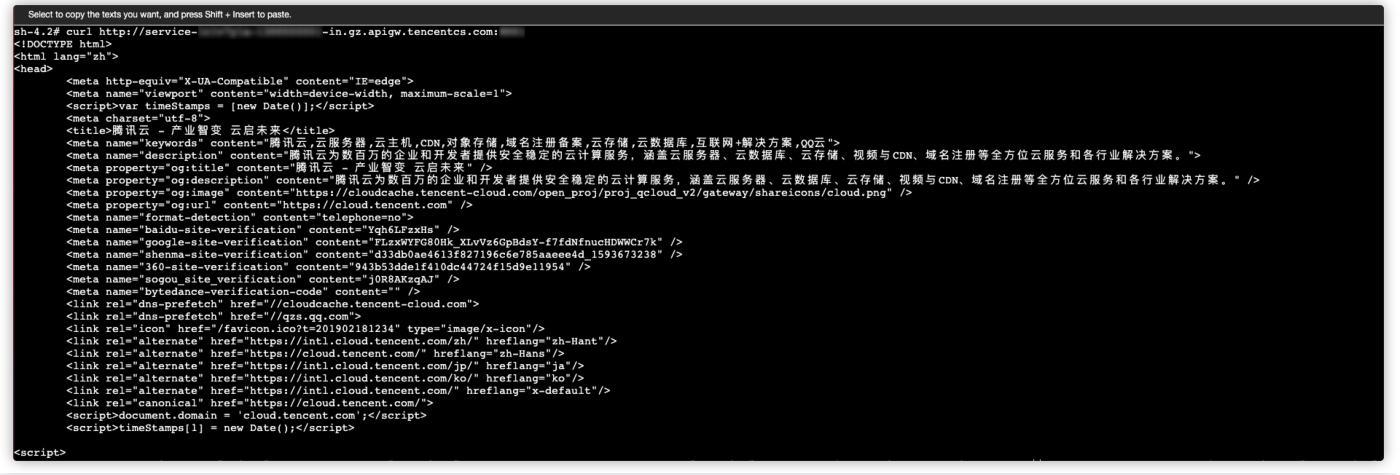
5. 在后端配置中，选择 公网URL/IP，配置您需要访问的公网域名以及路径（此处以腾讯云官网为例），单击**下一步**。

6. 设置应用的返回类型，此处为 HTML，RESTful 服务可选择为 JSON，单击完成。发布服务。

步骤2：验证公网请求连通性

1. 前往 API 网关服务基础配置页面，复制服务的内网 VPC 访问地址。

2. 打开部署好的弹性微服务应用页面，进入应用实例的 webshell，访问 API 网关内网 VPC 访问地址验证网络连通性。



弹性微服务应用运行失败故障排查指南

最近更新时间：2022-11-18 16:51:41

当弹性微服务应用处于运行失败状态时，说明至少有一个实例不处于 Running 状态，本文将列举几个常见的实例错误状态，并指导如何进行对应的故障排查。

实例错误状态

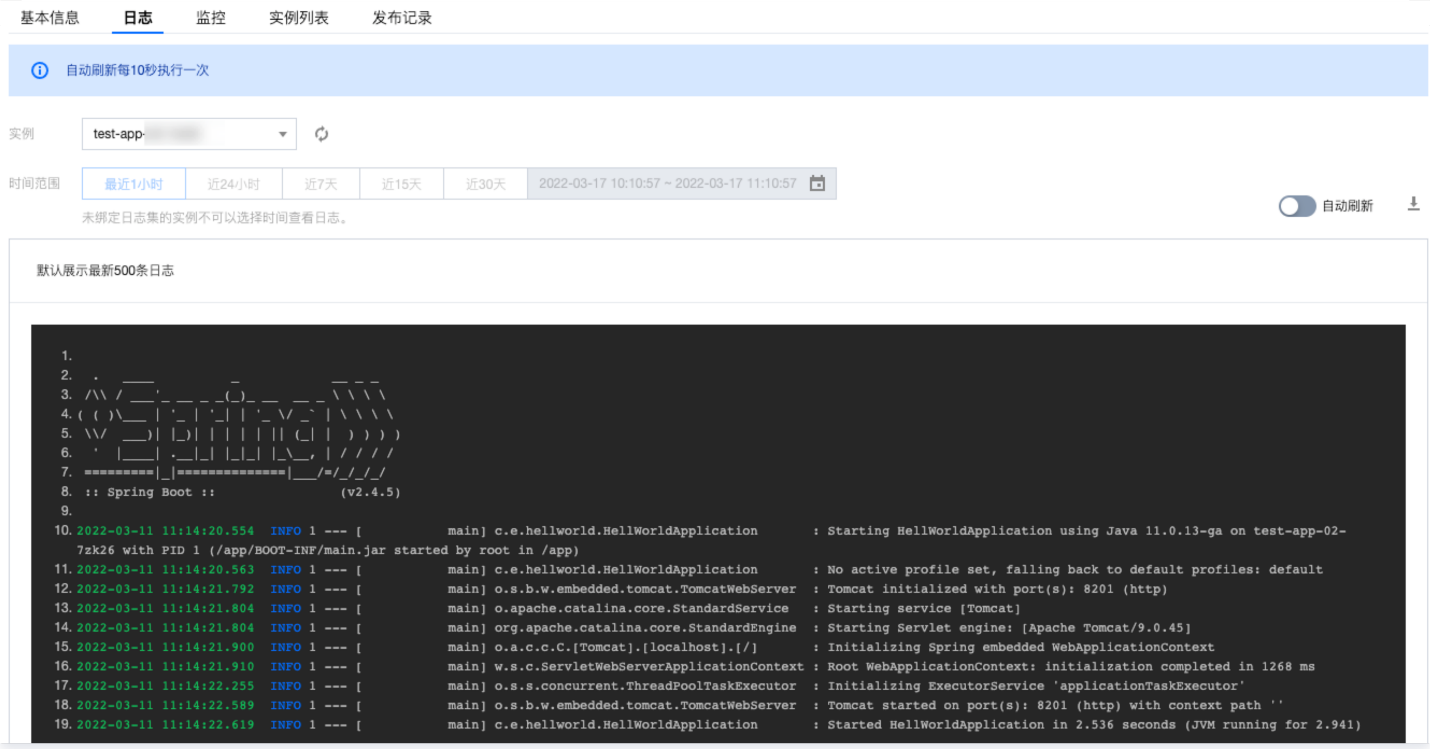
CrashLoopBackOff

状态说明

实例中应用程序的运行有问题，容器启动/运行失败。

解决方案

查看实例的日志，通过**日志内容**排查问题。



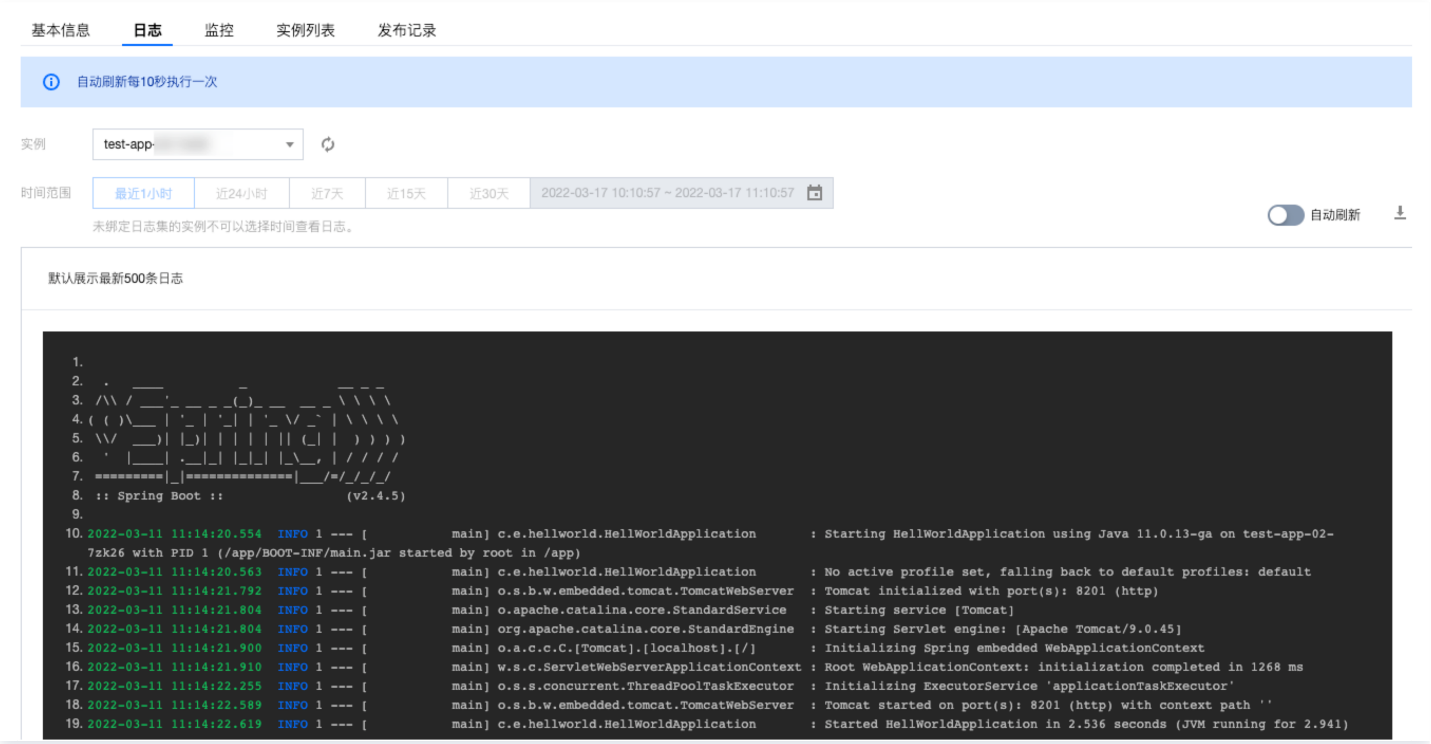
Error

状态说明

与 CrashLoopBackOff 类似，说明实例中应用程序的运行有问题，容器启动/运行失败。

解决方案

查看实例的日志，通过**日志内容**排查问题。



Running Unhealthy：Readiness probe failed

状态说明

应用配置的就绪健康检查失败。

解决方案

请在**应用部署** > **健康检查**页面中，检查应用的**就绪检查**配置是否正确。

☑ 就绪检查（Readiness）检查容器是否就绪，不就绪则将停止转发流量到当前实例，实例状态显示为「更新中」

检查方法 HTTP请求检查

检查协议 HTTP

请求路径 请输入请求的URL路径，如："/health"

端口 请输入请求端口

端口范围：1~65535，支持使用端口名

启动延时 10 秒

应用启动多久之后开始就绪检查，默认10秒

响应超时 3 秒

每次就绪检查超时时间，默认为3秒

检查周期 30 秒

每次就绪检查的时间间隔，默认为30秒

Running Unhealthy：Liveness probe failed

状态说明

应用配置的存活健康检查失败。

解决方案

请在**应用部署** > **健康检查**页面中，检查应用的**存活检查**配置是否正确。

Running Unhealthy：Readiness check failed according to l4 listener: xxx of CLB xxx. Service: xxx

状态说明

应用配置的访问配置无法连通。

解决方案

请在**应用详情** > **基本信息** > **访问配置**页面中，检查应用配置的**访问配置**的端口和协议是否正确。

PostStartHookError

状态说明

为应用配置的 PostStart 运行失败。

解决方案

请在**应用部署** > **启停管理**页面中，检查应用配置的 **PostStart** 是否能正常运行。

ContainerCreating

状态说明

实例容器创建失败。

解决方案

请在**应用部署** > **持久化存储**页面中，检查应用是否挂载了不存在的数据卷。

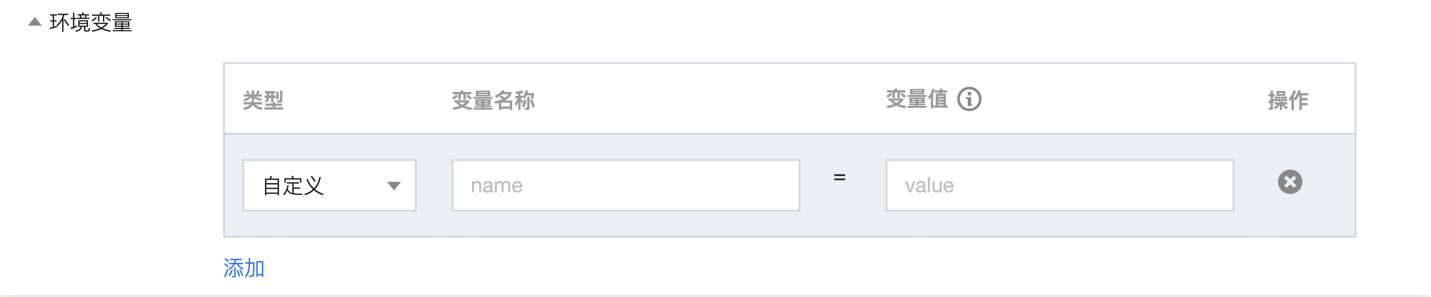
CreateContainerConfigError

状态说明

实例容器配置失败。

解决方案

请在**应用部署** > **环境变量**页面中，检查应用是否引用了不存在的配置。



ImagePullBackOff

状态说明

实例镜像拉取失败。

解决方案

前往 TCR控制台 检查应用使用的镜像是否存在或被误删除。

使用 API 网关快速访问弹性微服务应用

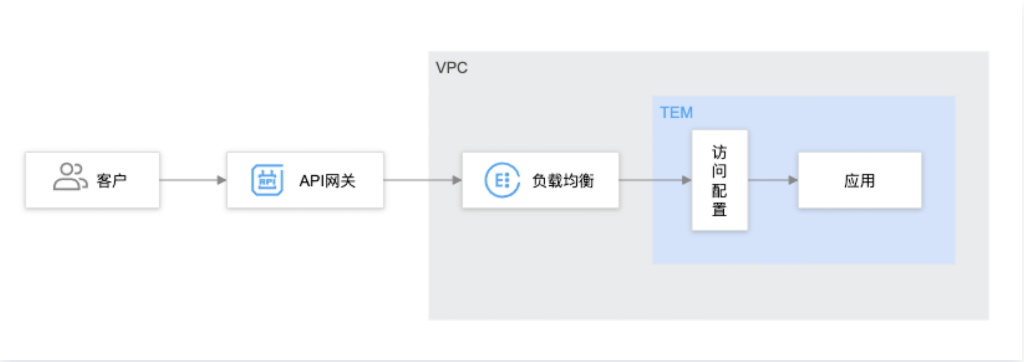
最近更新时间：2024-02-02 16:01:11

API 网关介绍

腾讯云 API 网关（API Gateway）是腾讯云推出的一种 API 托管服务，能提供 API 的完整生命周期管理，包括创建、维护、发布、运行、下线等。详细介绍请参见 API 网关官网文档 。

操作场景

本文档主要介绍如何快速使用腾讯云 API 网关访问弹性微服务应用并管理弹性微服务应用的 API。使用 API 网关和弹性微服务结合，可使弹性微服务的用户享受到 API 网关提供的限流、认证、缓存等高级能力，助力业务获得成功。



前提条件

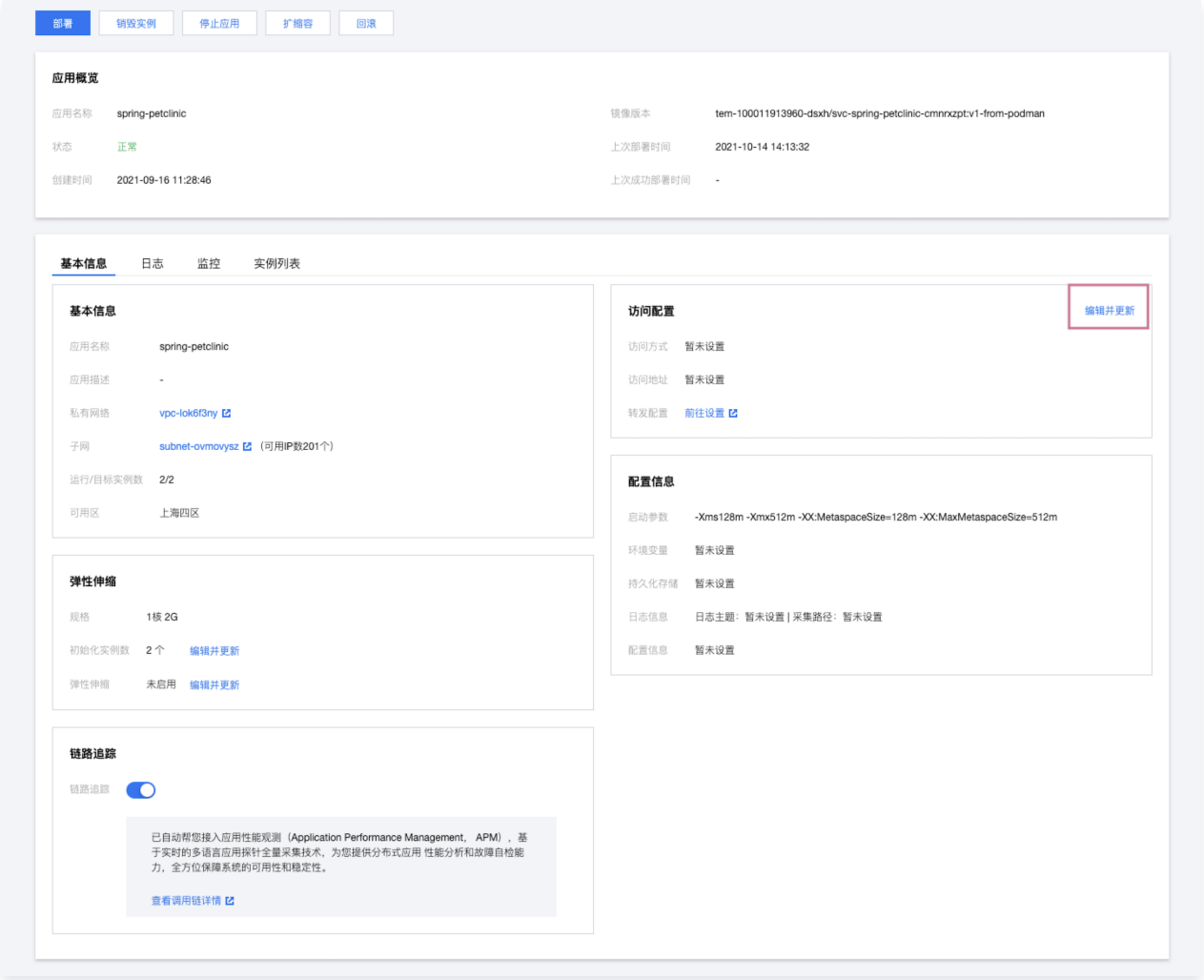
请登录 弹性微服务弹性微服务控制台，先完成 环境创建 和 创建并部署应用 。

操作步骤

步骤1：为弹性微服务应用配置 VPC 内网访问

1. 登录 弹性微服务控制台，在左侧导航栏单击**应用管理**，单击您想要配置的应用进入应用详情页。

2. 单击访问配置栏的**编辑并更新**，进入应用访问配置页。



3. 选择 VPC 内网访问（四层转发），选择子网、协议、容器端口和应用监听端口，并单击**提交**。此时弹性微服务会为您自动创建四层转发的 VPC 内网应用型 CLB。

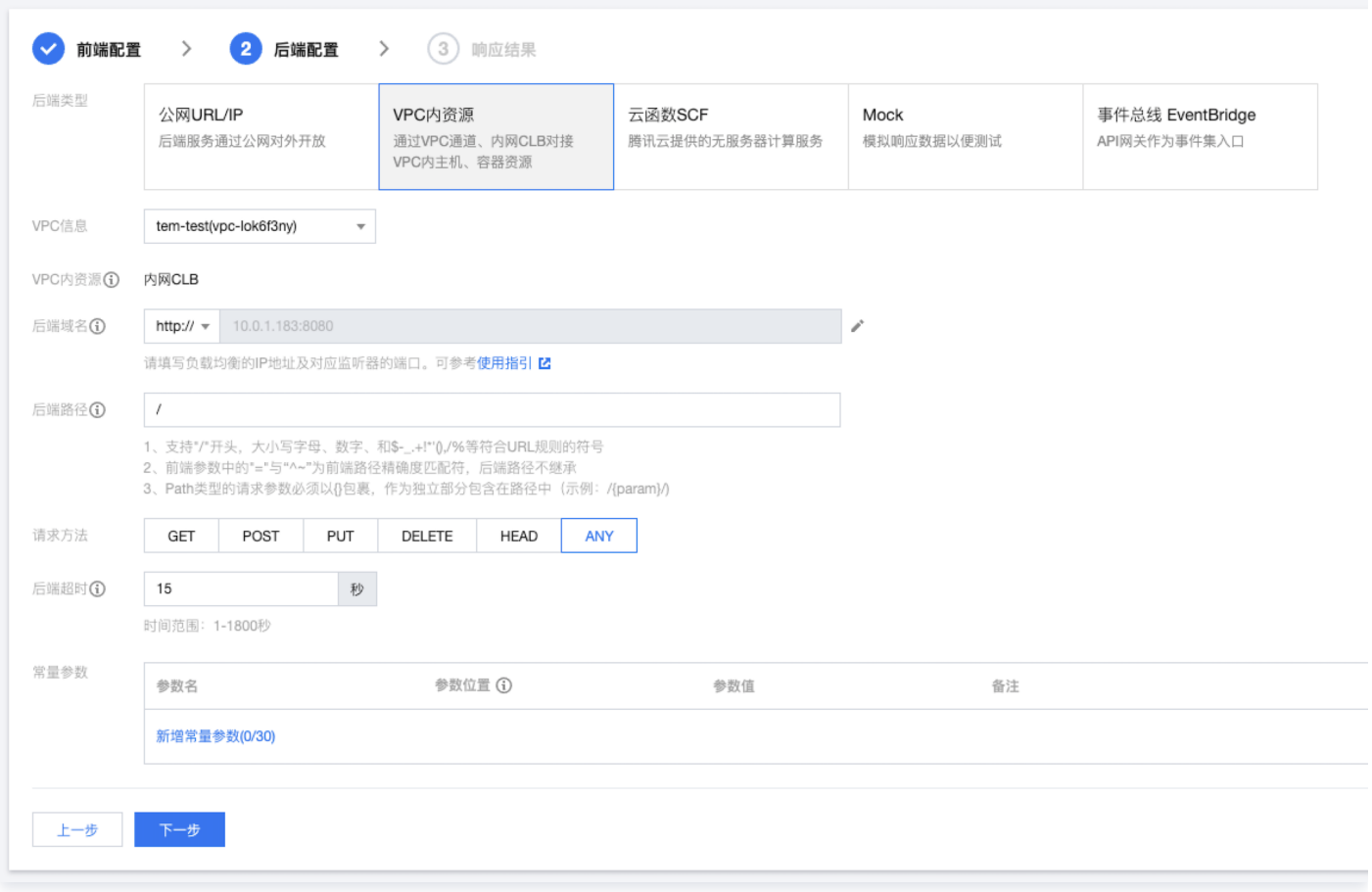
步骤2：创建 API 网关服务并绑定弹性微服务应用

1. 登录 API 网关控制台，在左侧导航栏，单击**服务**，进入服务列表页。
2. 选择与部署弹性微服务应用相同的地域，单击页面左上角的**新建**，新建一个服务。
   新建服务时，前端类型可选择 HTTP、HTTPS、HTTP &HTTPS 任一种，访问模式选择可以选择 VPC 内网和公网，实例类型选择共享性、专享型。

> **说明**
> 关于实例类型的选择，请参考 实例选择指南 。

3. 单击 API 网关服务 ID 进入 API 管理页面。单击**新建 API**。
4. 在前端配置中填写 API 名称，前端类型选择 HTTP&HTTPS，路径为“/”，请求方法选择 ANY 以包含所有请求，鉴权类型选择“免认证”，单击**下一步**。

5. 在后端配置中，选择 VPC 内资源，选择弹性微服务应用部署环境所在的 VPC。设置后端域名，选择弹性微服务应用自动创建的 CLB（名字为“cls-xxxdefault{弹性微服务应用名}”），选择相应的监听器（即上一步中所设置的端口映射），填写后端地址为“/”，完成 API 的创建。

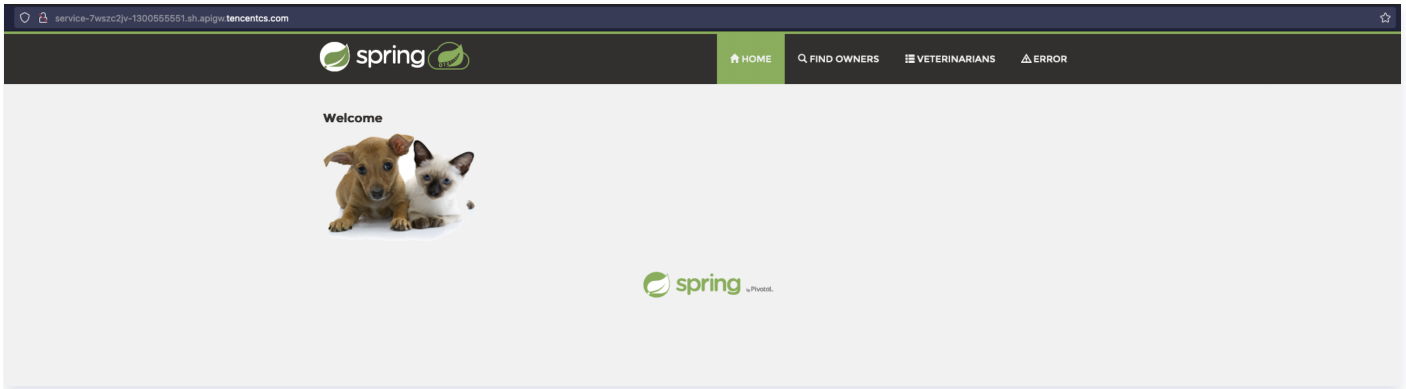


其中，后端域名的设置如下：

6. 此时您可看到您所配置的 API。并可以通过 API 网关提供的默认域名访问您的弹性微服务应用。

步骤3：通过 API 网关访问弹性微服务应用

访问 步骤2 中创建的 API 网关 API，即可通过 API 网关访问到弹性微服务应用。



注意事项

- 为保证应用无侵入的接入 API 网关，我们建议一个 API 网关服务只绑定一个弹性微服务应用，前端地址和后端地址保持一致，同为“/”可以拦截所有 API，您也可以在服务中为您应用的某些 API 进行单独的配置。
- 您可以参考 API 网关插件使用文档，为后端对接弹性微服务的 API 网关 API 绑定插件，以享受 API 网关提供的高级功能。

版权所有：腾讯云计算（北京）有限责任公司

Java 应用调优实践教程

最近更新时间：2024-06-14 18:19:51

优化容器镜像

通过优化容器镜像，您可以缩短加载时间和启动时间。您可以通过以下方式优化镜像：

- 尽可能减小化容器镜像大小
- 避免使用嵌套 jar 包
- 使用弹性微服务 Jar / War 方式部署

弹性微服务 Jar / War 方式部署默认提供 Jar 包镜像构建的实践教程，弹性微服务默认提供能充分利用构建缓存的构建流程，使用新一代构建利器 Buildkit 进行高速构建，构建速度优化50%以上，并且整个构建流程可追溯，构建日志可查，简单高效。

设置应用加速

使用弹性微服务 Jar / War 方式部署，并选择 KONA JDK 11/Open JDK 11 运行环境，弹性微服务将默认开启应用加速功能，并且默认支持 SpringBoot 应用零改造加速。弹性微服务增强了 Open JDK 中的 AppCDS 特性，您无需改造原有的 SpringBoot 嵌套 Jar 包结构，弹性微服务将直接提供 Java 应用加速的实践教程，实例扩容时的启动时间将缩短至10%～40%。

JVM 参数优化

使用可以感知容器内存资源的 JDK

在虚拟机和物理机中，对于 CPU 和内存分配，JVM 会从常见位置（例如，Linux 中的 `/proc/cpuinfo` 和 `/proc/meminfo` ）查找其可以使用的 CPU 和内存。但是，在容器中运行时，CPU 和内存限制条件存储在 `/proc/cgroups/...` 中。较旧版本的 JDK 会继续在 `/proc` （而不是 `/proc/cgroups` ）中查找，这可能会导致 CPU 和内存用量超出分配的上限，并因此引发多种严重的问题：

- 线程过多，因为线程池大小由 `Runtime.availableProcessors()` 配置。
- JVM的对内存使用超出容器内存上限。并导致容器被 OOMKilled。

JDK 8u131 首先实现了 `UseCGroupMemoryLimitForHeap` 的参数。但这个参数存在缺陷，为应用添加 `UnlockExperimentalVMOptions` 和 `UseCGroupMemoryLimitForHeap` 参数后，JVM 确实可以感知到容器内存，并控制应用的实际堆大小。但是这并没有充分利用我们为容器分配的内存。
因此 JVM 提供 `-XX:MaxRAMFraction` 标志来帮助更好的计算堆大小， `MaxRAMFraction` 默认值是4（即除以4），但它是一个分数，而不是一个百分比，因此很难设置一个能有效利用可用内存的值。
JDK 10 附带了对容器环境的更好支持。如果在 Linux 容器中运行 Java 应用程序，JVM 将使用 `UseContainerSupport` 选项自动检测内存限制。然后，通过 `InitialRAMPercentage` 、 `MaxRAMPercentage` 和 `MinRAMPercentage` 来进行对内存控制。这时，我们使用的是百分比而不是分数，这将更加准确。
默认情况下， `UseContainerSupport` 参数是激活的， `MaxRAMPercentage` 是25%， `MinRAMPercentage` 是50%。
需要注意的是，这里 `MinRAMPercentage` 并不是用来设置堆大小的最小值，而是仅当物理服务器（或容器）中的总可用内存小于250MB时，JVM 将用此参数来限制堆的大小。
同理， `MaxRAMPercentage` 是当物理服务器（或容器）中的总可用内存大小超过250MB时，JVM 将用此参数来限制堆的大小。
这几个参数已经向下移植到 JDK 8u191。UseContainerSupport 默认情况下是激活的。我们可以设置 `-XX:InitialRAMPercentage=50.0 -XX:MaxRAMPercentage=80.0` 来 JVM 感知并充分利用容器的可用内存。需要注意的是，在指定 `-Xms -Xmx` 时， `InitialRAMPercentage` 和 `MaxRAMPercentage` 将会失效。

关闭优化编译器

默认情况下，JVM 有多个阶段的 JIT 编译。虽然这些阶段可以逐渐提高应用的效率，但它们也会增加内存使用的开销，并增加启动时间。
对于短期运行的云原生应用，可以考虑使用以下参数来关闭优化阶段，以牺牲长期运行效率来换取更短的启动时间。

```
JAVA_TOOL_OPTIONS="-XX:+TieredCompilation -XX:TieredStopAtLevel=1"
```

关闭类验证

当 JVM 将类加载到内存中以供执行时，它会验证该类未被篡改并且没有恶意修改或损坏。但在云原生环境，CI/CD 流水线通常也由云原生平台提供，这表示我们的应用的编译和部署是可信的，因此我们应该考虑使用以下参数关闭验证。如果在启动时加载大量类，则关闭验证可能会提高启动速度。

```
JAVA_TOOL_OPTIONS="-noverify"
```

减小线程栈大小

大多数 Java Web 应用都是基于每个连接一个线程的模式。每个 Java 线程都会消耗本机内存（而不是堆内存）。这称为线程栈，并且每个线程默认为1MB。如果您的应用处理100个并发请求，则它可能至少有100个线程，这相当于使用了100MB的线程栈空间。该内存不计入堆大小。我们可以使用以下参数来减小线程栈大小。

```
JAVA_TOOL_OPTIONS="-Xss256k"
```

需要注意如果减小得太多，则将出现 `java.lang.StackOverflowError` 。您可以对应用进行分析，并找到要配置的最佳线程栈大小。

Spring boot 应用优化

使用 Spring Boot 2.2 或更高版本

从 2.2 版开始，Spring Boot 已针对启动速度进行了大量优化。如果您使用的是低于 2.2 版的 Spring Boot，请考虑升级或 手动进行优化 。

使用延迟初始化

在 Spring Boot 2.2 及更高版本中，您可以开启全局延迟初始化，以提高启动速度，但代价是第一个请求的延迟时间可能变长，因为需要等待组件首次初始化。
您可以在 `application.properties` 中开启延迟初始化：

```
spring.main.lazy-initialization=true
```

或者，使用以下环境变量：

```
SPRING_MAIN_LAZY_INITIALIZATIION=true
```

Java 8 迁移至 Java 11 实践教程

最近更新时间：2022-02-23 09:37:30

自 Java 9 发布以来，Java 已经在诸多方面进行了显著的改进和增强，并伴随着一些针对 API 的修改，这其中的许多功能都可以改善您应用的启动速度、性能和内存占用。

Java 8 与 Java 11 之间的显著更改

模块系统

模块系统 JSR 376 自 Java 9 引入 Java，它解决了在大型应用程序中，类路径的混乱、配置复杂和难以有效的封装问题 。
模块是 Java 类和接口以及相关资源的集合。模块可以定制应用程序的运行时配置。它占用空间更小，并允许使用 jlink 将应用链接到自定义运行时以进行部署。较小的占用空间在微服务架构中十分实用。JVM 在加载模块中的类时，也会比直接从类路径中加载时更快。
模块需要明确声明它导出哪些包以及它需要哪些组件，并且通过限制私有模块的反射访问来实现强封装。这种封装级别使应用更安全且更易于维护。
对于从 Java 8 过度来的开发者来说，模块并不是必选的，应用程序可以继续使用类路径在 Java 11 上运行。
您可以参见 The State of the Module System 来了解模块系统的工作方式。

JVM 的分析诊断工具

Java Flight Recorder 和 Java Mission Control

Java Flight Recorder (JFR) JEP 328 从正在运行的 Java 应用程序收集诊断和分析数据。JFR 对正在运行的 Java 应用程序几乎没有影响。开发者可以使用 Java Mission Control (JMC) 和其他工具分析收集到的数据。 JFR 和 JMC 在 Java 8 中是商业功能，但它们在 Java 11 中都是开源的。

JVM 日志系统

Java 11 为 JVM 的所有组件提供了一个通用的日志记录系统 JEP 158 。这个统一的日志系统允许用户定义想要记录的组件，以及记录到什么级别。这种细粒度的日志能帮助开发者在 JVM 崩溃时进行根因分析，以及诊断生产环境中的性能问题。

低开销的堆分析

Java 虚拟机工具接口 (JVMTI) 中添加了新的 API，用于对 Java 堆分配进行采样 JEP 331 。采样具有低开销的特性。

垃圾回收

Java 11 中提供了以下垃圾收集器：串行（Serial）、并行（Parallel）、G1（Garbage-First）和 Epsilon。Java 11 中的默认垃圾收集器是 G1。

- G1 的目标是在延迟和吞吐量之间取得平衡。G1 旨在避免 Full GC，但是当并发收集不能足够快地回收内存时，仍然会产生 Full GC。
- 并行GC 是 Java 8 中的默认收集器，它是一种吞吐量优先的收集器，它使用多个线程来加速垃圾收集。
- Epsilon 垃圾收集器处理分配但不回收任何内存。当堆内存耗尽时，JVM 将关闭。Epsilon 对于短期服务和无垃圾的应用程序会很有用。

除此之外，Java 11 还提供其他三种垃圾回收器：

- ZGC 是一个并发、低延迟的收集器，它试图将暂停时间保持在 10 毫秒以下。ZGC 作为 Java 11 中的一项实验性功能提供。
- Shenandoah 是一种低暂停收集器，它会与正在运行的 Java 程序同时执行，并以此来减少 GC 暂停时间。Shenandoah 是 Java 12 中的一项实验性功能，但已经移植到 Java 11。
- CMS 虽然在Java 11 中仍然可用，但自 Java 9 以来已被标记为弃用。

容器环境的改进

在 Java 10 之前，JVM 无法识别在容器上设置的内存和 CPU 限制。例如，在 Java 8 中，JVM 默认最大堆为底层主机物理内存的 1/4。从 Java 10 开始，JVM 使用 cgroups 来设置内存和 CPU 限制。例如，默认的最大堆是容器内存的 1/4。
此外，Java 10 中还提供了新的 JVM 参数，以便 Docker 容器用户可以细粒度地控制将用于 Java 堆的系统内存量。

> **说明**

> 从 jdk8u191 开始，大部分 cgroup 支持工作都被向后移植到 Java 8。

从 Java 8 迁移到 Java 11

在应用从 Java 8 迁移到 Java 11 的过程中，并不存在万能的解决方案，开发者可能会面临的潜在问题包括：已删除的 API、已弃用的包、内部 API 的使用、类加载器的更改以及垃圾回收器的更改。

尝试直接编译运行

一般来说，最简单的方法是尝试在不重新编译的情况下，直接在 Java 11 上运行使用 Java 8 编译的应用，或者先使用 Java 11 进行编译后，再运行。如果目标是让应用程序尽快启动并运行，那么这种简单的尝试通常是最好的方法。

其他工具

Java 11 提供了两个工具，jdeprscan 和 jdeps，可用于发现潜在问题。这些工具可以针对现有的类文件或 jar 包运行，您无需重新编译即可使用。

jdeprscan

jdeprscan 用于查找程序中是否有使用已弃用或已删除的 API 。使用已弃用的 API 并不会阻塞您的迁移，但您仍需要注意，因为他们有可能在未来的版本中被删除。
若要使用 jdeprscan，最简单的方法是提供一个现成的 jar 包，您可以为其指定目录或是某个类名。 使用 `--release 11` 参数可获取使用已弃用 API 的最完整列表。例如 `jdeprscan --release 11 my-application.jar` 。
如果您遇到 `error: cannot find class XXX` 的错误，则需要优先检查此依赖类文件是否存在于 jar 包的类路径中。如此依赖类并非第三方依赖，则有可能是您使用了在 Java 11 中已被删除的 API。
运行 `jdeprscan --release 11 --list` 即可了解自 Java 8 后弃用的具体 API。 若要获取已删除 API 的列表，请运行 `jdeprscan --release 11 --list --for-removal` 。

jdeps

jdeps，是一个 Java 类依赖分析器。当此工具与 `--jdk-internals` 参数一起使用时，jdeps 会告诉您的哪个类依赖内部 API。我们也建议添加 `--multi-release 11` 参数来支持多版本构建的 jar 包。例如 `jdeps --jdk-internals --multi-release 11 --class-path log4j-core-2.13.0.jar my-application.jar` 。
您可以继续在 Java 11 中使用内部 API，但这种方式已经不在推荐。OpenJDK wiki Java Dependency Analysis Tool 推荐了一些常用 JDK 内部 API 的替代品。
您应该尽量避免使用来自 jdk.unsupported 中的任何 API 。尽管在替代的 API 可用之前，使用内部 API 仍将被支持，但在未来，它们有可能被彻底弃用或删除。JEP 260 提供了一些替代方案。
Gradle 和 Maven 都有 jdeps 和 jdeprscan 插件。我们建议将这些工具添加到您的构建脚本中。

工具	Gradle 插件	Maven 插件
jdeps	jdeps-gradle-插件	Apache Maven JDeps 插件
jdeprscan	jdeprscan-gradle-插件	Apache Maven JDeprScan 插件

jdeprscan 和 jdeps 无法检查使用反射访问的 API。因此，您需要在运行时检查您代码中的反射访问。

运行时检查

检查 JVM 参数

在 Java 11 上运行之前，请检查 JVM 参数。 使用已删除的 JVM 参数将导致 JVM 崩溃退出（ `Error: Could not create the Java Virtual Machine` ）。如果您开启了 GC 日志，则此检查尤其重要，因为 GC 日志与 Java 8 相比发生了巨大变化。您可以使用 JaCoLine 工具来检测 JVM 参数。

检查第三方依赖类库

您需要让您的所有第三方依赖类库都更新为支持 Java 11 的版本。对此，OpenJDK Quality Group 维护了一个 Quality Outreach wiki 页面，其中列出了许多免费开源软件 (FOSS) 项目针对 OpenJDK 版本的测试状态。

检查垃圾回收参数

并行垃圾回收器（Parallel GC）是 Java 8 中的默认 GC。自 Java 9 开始，默认的垃圾回收器更改为 G1GC。您需要检查您的垃圾回收参数是否正确设置。

类加载器注意事项

- 在 Java 11 中的类加载器层次结构发生了变化。 `SystemClassloader` （也称为 `AppClassloader` ）现在是一个内部类。强制转换成 `URLClassLoader` 会抛出 `ClassCastException` 异常。 Java 11 没有在运行时动态增加 classpath 的 API，但您仍然可以通过反射来完成获取。
- 在 Java 11 中， `BootstrapClassloader` 只加载核心模块。如果您创建一个没有父级加载器的 classloader，它可能无法找到所有平台类。在 Java 11 中，您需要传递 `ClassLoader.getPlatformClassLoader()` 作为其父级加载器。

语言环境数据更改

Java 11 中语言环境数据的默认源已随 JEP 252 更改为 Unicode 的通用区域资料库。这可能会对本地化产生影响。如有必要，设置系统属性 `java.locale.providers=COMPAT,SPI` 以恢复为 Java 8 语言环境行为。

一些常见的问题

Unrecognized options

如果 JVM 参数已被删除，应用程序将打印 `Unrecognized option:` 或 `Unrecognized VM option` 。无法识别的参数将导致 JVM 崩溃退出（ `Error: Could not create the Java Virtual Machine` ）。已弃用但未删除的选项将产生 JVM 警告（ `VM Warning: Option <option> was deprecated` ）。
通常情况下，您需要删除这些不能识别的 JVM 参数。GC 日志参数除外。GC 日志在 jep 271 中重新实现。请参见 oracle 官方文档 进行重新配置。

WARNING: An illegal reflective access operation has occurred

当 Java 代码使用反射访问 JDK 内部 API 时，运行时会发出非法反射访问警告。

java.lang.reflect.InaccessibleObjectException

此异常表明您正在试图通过 `setAccessible(true)` 的方式来反射获取一个包/模块中私有的类的字段或方法。使用 `--add-opens` 参数可让您的代码访问包/模块的非公共成员。

java.lang.NoClassDefFoundError

如果应用程序在 Java 8 上正常运行，但在Java 11中抛出 `java.lang.NoClassDefFoundError` 或 `java.lang.ClassNotFoundException` ，那么很可能应用程序正在使用来自 Java EE 或 CORBA 模块的包。这些模块在 Java 9 中被弃用并在 Java 11 中被删除 jep 320 。
要解决此问题，请向您的项目添加运行时依赖。

移除模块	受影响的包	建议的依赖
JAX-WS	java.xml.ws	JAX WS RI 运行时
JAXB	java.xml.bind	JAXB 运行时
JAV	java.activation	JavaBeans (TM) 激活框架
注解	java.xml.ws.annotation	Javax 注解 API
CORBA	java.corba	GlassFish CORBA ORB
JTA	java.transaction	Java 事务 API

UnsupportedClassVersionError

此异常意味着您正在尝试在早期的 Java 版本上运行使用更高版本的 Java 编译的代码。例如，您在 Java 11 上运行一个使用 JDK 13 编译的 jar。

Java版本	类文件版本
8	52
9	53
10	54
11	55
12	56
13	57